Data Structures and Algorithms Board Infinity
A Training Report

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology (Computer Science and Engineering)

Submitted to:

Lovely Professional University, Phagwara, Punjab

From 28th May 2024 to 20st July 2024

Submitted by:

Name of Student**: Annareddy Sanjana**

Registration Number**: 12220788**

LIST OF CONTENTS

To whom so ever it may concern

I, **ANNAREDDY SANJANA**, **12220788**, hereby declare that the work done by me on **"Data Structures and Algorithms"** from **28th May 2024 to 20th July 2024**, is a record of original work for the partial fulfilment of the requirements for the award of the degree, **Bachelor of Technology**.

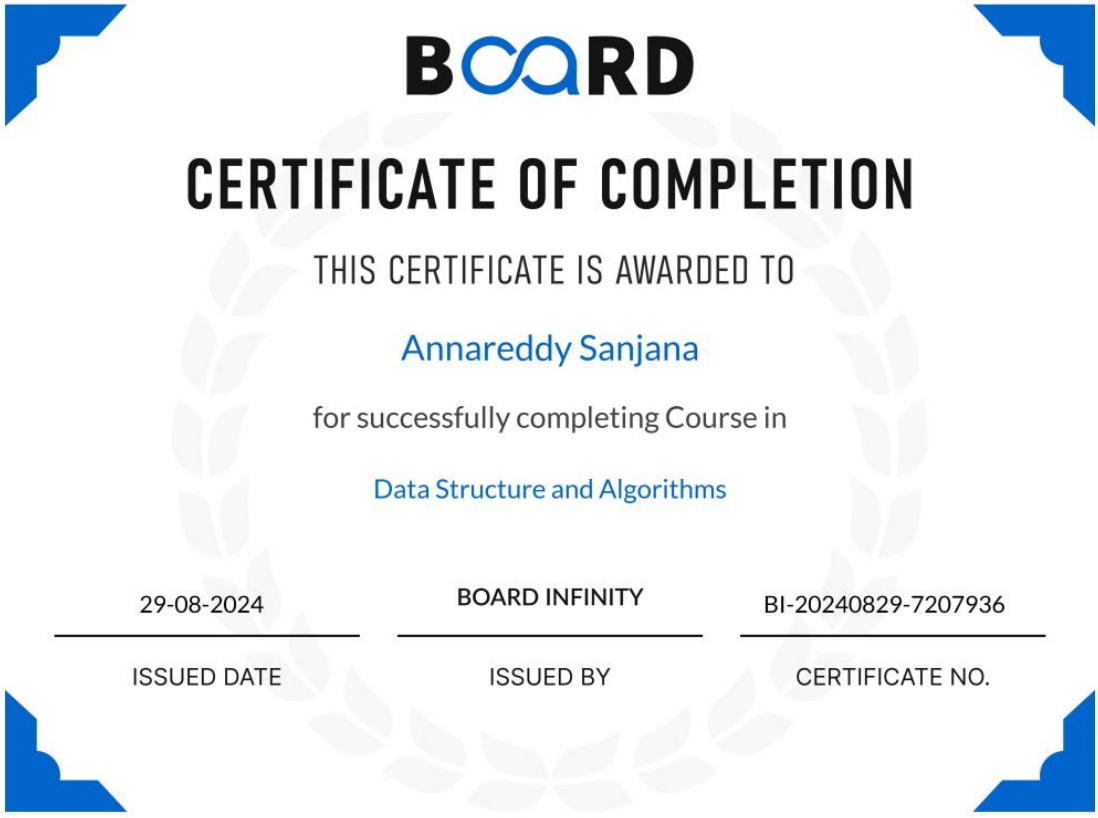
Name of the Student:

Annareddy Sanjana

Registration Number: 12220788

Dated: 25th August 2024

SUMMER TRAINING CERTIFICATE



BOARD

CERTIFICATE OF COMPLETION

THIS CERTIFICATE IS AWARDED TO

Annareddy Sanjana

for successfully completing Course in

Data Structure and Algorithms

29-08-2024	BOARD INFINITY	BI-20240829-7207936
ISSUED DATE	ISSUED BY	CERTIFICATE NO.

ACKNOWLEDGEMENT

First and foremost, I am deeply grateful for the opportunity to expand my knowledge and skills in a new technology. I would like to extend my sincere thanks to the instructor of the DSA course from Board infinity, whose guidance has been valuable in facilitating my learning journey from home.

I am also thankful to my college, Lovely Professional University, for providing access to such an enriching course that not only enhanced my programming abilities but also introduced me to new technologies.

I would like to express my heartfelt appreciation to my parents and friends for their unwavering support, valuable advice, and encouragement in choosing this course. Lastly, my gratitude extends to my classmates, whose collaboration and assistance have been greatly appreciated.

LIST OF ABBREVIATION

DSA – Data Structures and Algorithms

INTRODUCTION OF THE PROJECT UNDERTAKEN

➢ **What is a Data Structure?** A data structure is a method of organizing and managing data so that operations can be performed efficiently. It involves arranging data elements based on specific relationships to improve organization and storage. For instance, if we have data with a player's name "Vishnu" and age 20, "Vishnu" is a String type, while 20 is an Integer type.

➢ **What is an Algorithm?** An algorithm is a set of step-bystep instructions or logic, arranged in a specific order, to achieve a particular task. It is the fundamental solution to a problem, presented either as a high-level description in pseudocode or through a flowchart, but it is not the full code or program.

➢ This course provided a comprehensive learning experience, guiding me through Data Structures and Algorithms from the basics to advanced concepts. The curriculum is structured into 7 weeks, allowing participants to work through practice problems and assessments at their own pace. The course offers a variety of programming challenges that are invaluable in preparing for interviews with leading companies like Microsoft, Amazon, Adobe, and others.

TECHNOLOGY LEARNT

Analysis of Algorithms

- **Analysis of Algorithm:** This section focused on understanding background analysis through programs and their functions.

- **Order of Growth:** I learned about the mathematical representation of growth analysis using limits and functions, including a straightforward approach to determining the order of growth.

- **Asymptotic Notations:** This covered the best, average, and worst-case scenarios, explained through example programs.

- **Big O Notation:** The concept was explained both graphically and mathematically, with calculations and applications demonstrated using Linear Search.

- **Omega Notation:** I explored this notation through graphical and mathematical explanations, along with calculations.

- **Theta Notation:** The Theta notation was explained with graphical and mathematical insights, including detailed calculations.

- **Recursion Analysis:** This included various calculations using the Recursion Tree method.

- **Space Complexity:** I explored basic programs, auxiliary space, and analyzed the space complexity of recursive functions and the Fibonacci sequence.

Mathematics

- **Digit Counting:** Techniques for finding the number of digits in a number.

- **Progressions:** Arithmetic and geometric progressions were covered.

Statistical Measures: Concepts of mean and median.

Prime Numbers: Understanding prime numbers.

- **LCM and HCF:** Learning about Least Common Multiple (LCM) and Highest Common Factor (HCF).

Factorials: Calculating factorials.

- **Permutations and Combinations:** Exploring permutations and combinations.

- **Modular Arithmetic:** Learning the basics of modular arithmetic.

Bitwise Operations

Bitwise Operators in C++: I explored the operations of AND, OR, XOR, Left Shift, Right Shift, and Bitwise NOT in C++.

Recursion

- **Introduction:** An introduction to recursion, including its applications.

- **Writing Base Cases:** Learning to write base cases in recursion through examples like factorial and N-th Fibonacci number.

Arrays

- **Introduction and Benefits:** An overview of arrays and their advantages.

- **Types of Arrays:** Covered fixed-sized and dynamic-sized arrays.

- **Array Operations:** Studied operations like searching, insertion, deletion, comparison with other data structures, and reversing arrays with complexity analysis.

Searching

- **Binary Search:** I learned both iterative and recursive approaches to binary search, along with associated problems.

- **Two Pointer Approach:** Explored problems that utilize the two-pointer approach.

Sorting

- **STL Sort in C++:** Implementing the **sort()** function in arrays and vectors in C++ with a focus on time complexities.
- **Stability in Sorting Algorithms:** Analyzed stable and unstable sorting algorithms with examples.
- **Sorting Algorithms:** I implemented and analyzed sorting algorithms like Insertion Sort, Merge Sort, and Quick Sort (including Lomuto and Hoare partitioning methods), along with their time and space complexities, pivot selection, and worst-case scenarios.
- **Sorting Algorithms Overview:** A broad overview of sorting algorithms.

Hashing

- **Introduction to Hashing:** Time complexity analysis and applications of hashing were introduced.
- **Hash Functions:** I learned about Direct Address Tables and various hashing functions.
- **Collision Handling:** Discussed different collision handling techniques, including chaining and open addressing, with implementations.
- **Hashing in C++:** Explored **unordered_set** and **unordered_map** in C++, and **HashSet** and **HashMap** in Java.

Strings

- **String Data Structure:** Studied string data structures in C++ .
- **String Algorithms:** Implemented string algorithms like Rabin Karp and KMP.

Linked Lists

- **Introduction:** An introduction to linked lists with implementations in C++ and Java.
- **Types of Linked Lists:** Explored doubly linked lists and circular linked lists.
- **Loop Problems:** Covered problems like detecting loops using Floyd's cycle detection algorithm, and detecting and removing loops in linked lists.

Stacks

- **Understanding Stacks:** An overview of the stack data structure and its applications.

- **Stack Implementation:** Implementation of stacks using arrays and linked lists in C++ .

Queues

- **Introduction to Queues:** Basics and applications of queues.

- **Queue Implementation:** Implemented queues using arrays and linked lists in C++ STL, including stack-based queue implementation.

Deques

- **Introduction to Deques:** Studied deque data structure and its applications.

- **Deque Implementation:** Implemented deques in C++ STL .

- **Problem Solving:** Tackled problems like finding the maximum of all subarrays of size k, and designing a data structure with min-max operations.

Trees

- **Introduction to Trees:** Covered the basics of trees, including binary trees and their applications.

- **Tree Traversals:** Implemented various tree traversal techniques like Inorder, Preorder, Postorder, Level Order (line by line).

Binary Search Trees (BST)

- **Introduction to BST:** An overview of Binary Search Trees, including background, introduction, and applications.

- **BST Operations:** Implemented search, insertion, deletion, and floor operations in BST, along with exploring self-balancing BSTs.

Heaps

- **Introduction to Heaps:** Basics and implementation of heaps.

- **Binary Heap Operations:** Implemented binary heap operations like insertion, heapify, extract, decrease key, delete, and building a heap.

- **Heap Sort:** Studied heap sort, and priority queues in C++.

Graphs

- **Introduction to Graphs:** Basics of graph data structures and their representation using adjacency matrix and adjacency list in C++ and Java.

- **Graph Traversal:** Implemented Breadth-First Search (BFS) and Depth-First Search (DFS) along with their applications.

- **Shortest Path Algorithms:** Studied algorithms for finding the shortest path in Directed Acyclic Graphs, Prim's Algorithm for Minimum Spanning Tree, Dijkstra's Shortest Path Algorithm, Bellman-Ford Algorithm.

➢ Enhanced my problem-solving abilities by tackling a variety of challenges to become a more proficient developer.

- **Practice Problems:** This track offers numerous essential practice problems that are crucial for mastering Data Structures and Algorithms.

➢ Sharpened my analytical skills in Data Structures, enabling me to utilize them more effectively.

➢ Successfully tackled problems commonly encountered in interviews with product-based companies.

➢ Competed in contests that mirror the coding rounds for Software Development Engineer (SDE) roles.

PROJECT

SIMPLE LIBRARY MANAGEMENT SYSTEM

Overview

This C++ program implements a **Library Management System** that allows users to manage a collection of books. The program uses a Book structure to store details about each book, including its ID, title, author, and whether it is currently issued. The Library class manages a collection of Book objects and provides various functionalities such as adding new books, searching for books by ID or title, issuing and returning books, listing all books, and deleting books from the library.

Code Implementation

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

struct Book {
    int id;
    string title;
    string author;
    bool isIssued;

    Book(int id, const string& title, const string& author)
        : id(id), title(title), author(author), isIssued(false) {}
};

class Library {
private:
    vector<Book> books;
public:
    void addBook(int id, const string& title, const string& author);
    void searchBookById(int id);
    void searchBookByTitle(const string& title);
    void issueBook(int id);
    void returnBook(int id);
    void listAllBooks();
    void deleteBook(int id);
};
```

```cpp
void Library::addBook(int id, const string& title, const string& author) {
    books.push_back(Book(id, title, author));
    cout << "Book added successfully.\n";
}

void Library::searchBookById(int id) {
    for (const auto& book : books) {
        if (book.id == id) {
            cout << "Book found"<<endl<< "ID: " << book.id << endl<< "Title: " << book.title <<
            endl<<"Author: " << book.author << endl<<"Status: " << (book.isIssued ? "Issued" : "Available") << '\n';
            return;
        }
    }
    cout << "Book not found.\n";
}

void Library::searchBookByTitle(const string& title) {
    for (const auto& book : books) {
        if (book.title == title) {
            cout << "Book found"<<endl<<"ID: " << book.id <<endl<< "Title: " << book.title << endl <<"Author: " <<
            book.author << endl<<"Status: " << (book.isIssued ? "Issued" : "Available") << '\n';
            return;
        }
    }
    cout << "Book not found.\n";
}
```

```cpp
void Library::issueBook(int id) {
    for (auto& book : books) {
        if (book.id == id) {
            if (!book.isIssued) {
                book.isIssued = true;
                cout << "Book issued successfully.\n";
            } else {
                cout << "Book is already issued.\n";
            }
            return;
        }
    }
    cout << "Book not found.\n";
}

void Library::returnBook(int id) {
    for (auto& book : books) {
        if (book.id == id) {
            if (book.isIssued) {
                book.isIssued = false;
                cout << "Book returned successfully.\n";
            } else {
                cout << "Book was not issued.\n";
            }
            return;
        }
    }
    cout << "Book not found.\n";
}
```

```cpp
void Library::listAllBooks() {
    sort(books.begin(), books.end(), [](const Book& a, const Book& b) {
        return a.id < b.id;
    });
    for (const auto& book : books) {
        cout << "ID: " << book.id <<endl<< "Title: " << book.title <<endl<< "Author: " << book.author
        <<endl<< "Status: " << (book.isIssued ? "Issued" : "Available") << '\n';
    }
}

void Library::deleteBook(int id) {
    auto it = remove_if(books.begin(), books.end(), [id](const Book& book) {
        return book.id == id;
    });
    if (it != books.end()) {
        books.erase(it, books.end());
        cout << "Book deleted successfully.\n";
    } else {
        cout << "Book not found.\n";
    }
}

```

```cpp
int main() {
    Library library;
    int choice, id;
    string title, author;

    while (true) {
        cout << "Library Management System\n";
        cout << "1. Add New Book\n2. Search Book by ID\n3. Search Book by Title\n4. Issue Book\n
        5. Return Book\n6. List All Books\n7. Delete Book\n8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter Book ID: ";
                cin >> id;
                cin.ignore(); // ignore newline character
                cout << "Enter Book Title: ";
                getline(cin, title);
                cout << "Enter Book Author: ";
                getline(cin, author);
                library.addBook(id, title, author);
                break;
            case 2:
                cout << "Enter Book ID: ";
                cin >> id;
                library.searchBookById(id);
                break;
```

```
138             case 3:
139                 cout << "Enter Book Title: ";
140                 cin.ignore();
141                 getline(cin, title);
142                 library.searchBookByTitle(title);
143                 break;
144             case 4:
145                 cout << "Enter Book ID to issue: ";
146                 cin >> id;
147                 library.issueBook(id);
148                 break;
149             case 5:
150                 cout << "Enter Book ID to return: ";
151                 cin >> id;
152                 library.returnBook(id);
153                 break;
154             case 6:
155                 library.listAllBooks();
156                 break;
157             case 7:
158                 cout << "Enter Book ID to delete: ";
159                 cin >> id;
160                 library.deleteBook(id);
161                 break;
162             case 8:
163                 return 0;
164             default:
165                 cout << "Invalid choice, please try again.\n";
166         }
167     } return 0; }
```

OUTPUT :

```
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 1
Enter Book ID: 1
Enter Book Title: pride and prejudice
Enter Book Author: jane austen
Book added successfully.
```

```
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 2
Enter Book ID: 1
Book found
ID: 1
Title: pride and prejudice
Author: jane austen
Status: Available
```

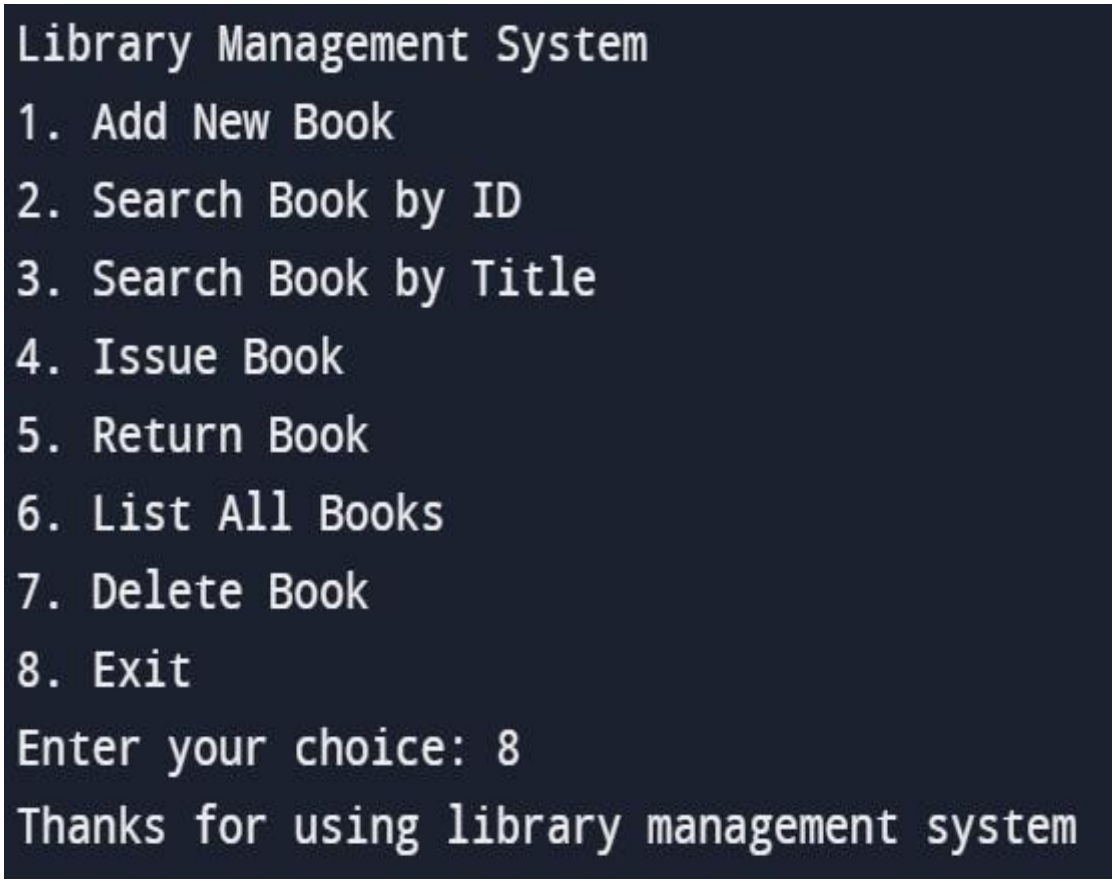
```
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 3
Enter Book Title: pride and prejudice
Book found
ID: 1
Title: pride and prejudice
Author: jane austen
Status: Available
```

```
Enter your choice: 4
Enter Book ID to issue: 1
Book issued successfully.
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 4
Enter Book ID to issue: 1
Book is already issued.
```

```
Enter your choice: 4
Enter Book ID to issue: 1
Book issued successfully.
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 5
Enter Book ID to return: 1
Book returned successfully.
```

```
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 6
ID: 1
Title: pride and prejudice
Author: jane austen
Status: Available
```

```
Library Management System
1. Add New Book
2. Search Book by ID
3. Search Book by Title
4. Issue Book
5. Return Book
6. List All Books
7. Delete Book
8. Exit
Enter your choice: 7
Enter Book ID to delete: 1
Book deleted successfully.
```



Functionality

The Library Management System project is a simple yet effective application designed to manage and organize a collection of books within a library. The system offers various features that assist in handling book records, issuing and returning books, and searching for specific titles or authors. The main features include:

- **Book Addition**: The system allows the librarian to add new books to the library's collection by providing a unique ID, title, and author name. Each new book entry is stored in the system, and it is initially marked as available for issue.
- **Search by ID**: Users can search for a specific book by its unique ID. The system retrieves and displays the book's details, including its title, author, and current availability status (whether issued or available).

- **Search by Title**: The system also enables searching for books by their title. It provides detailed information about

the book, similar to the search by ID feature, and helps users locate the book within the library's collection.

- **Book Issuing**: The librarian can issue a book to a user by entering the book's ID. The system checks the availability of the book and, if it is available, marks it as issued. If the book is already issued, the system informs the librarian to avoid double issuance.

- **Book Return**: When a book is returned, the librarian can update the book's status in the system by entering the book’s ID. This changes the status of the book back to available, making it ready for future issuance.

- **List All Books**: The system provides an organized list of all books currently in the library, sorted by their unique IDs. The list includes each book's ID, title, author, and availability status, giving the librarian a quick overview of the library’s inventory.

- **Book Deletion**: If a book is no longer part of the library's collection, the librarian can delete its record by entering the book's ID. The system ensures that the book is removed from the list, keeping the library's inventory up to date.

Overall, this Library Management System enhances the management and accessibility of the library’s resources, making it easier for librarians to track, issue, and maintain books efficiently.

REASON FOR CHOOSING DSA

- With rapid advancements in technology, programming has become an essential and highly sought-after skill for Software Developers. Everything from smart devices like

TVs and ACs to traffic lights relies on programming to execute user commands.

- **Efficiency is Key:** To remain irreplaceable, one must be efficient. Mastery of Data Structures and Algorithms is a hallmark of a skilled Software Developer. Technical interviews at leading tech companies such as Google, Facebook, Amazon, and Flipkart often focus heavily on a candidate's proficiency in these areas. This focus is due to the significant impact that Data Structures and Algorithms have on enhancing a developer's problem-solving capabilities.

- **Learning Method:** This course offered comprehensive video lectures on all topics, which suited my preferred learning style. While books and notes have their importance, video lectures facilitate faster understanding through practical involvement.

- **Extensive Practice:** The course included algorithmic coding problems with video explanations, providing a rich resource for hands-on learning.

- **Structured Learning:** The course was organized with track-based learning and weekly assessments to continually test and improve my skills.

LEARNING OUTCOMES

Programming fundamentally revolves around the use of data structures and algorithms. Data structures manage and store data, while algorithms solve problems by processing that data.

Data Structures and Algorithms (DSA) explore solutions to common problems in depth, providing insights into the efficiency of different approaches. Understanding DSA empowers you to select the most effective methods for any given task. For example, if you need to search for a specific record in a database of 30,000 entries, you could choose between methods like Linear Search and Binary Search. In this scenario, Binary Search would be the more efficient choice due to its ability to quickly narrow down the possibilities.

Knowing DSA allows you to tackle problems with greater efficiency, making your code more scalable, which is crucial because:

- Time is valuable.
- Memory is costly.

DSA is also essential for excelling in technical interviews at product-based companies. Just as we prefer someone who can complete a task quickly and efficiently in our daily lives, companies look for developers who can solve complex problems effectively and with minimal resources.

For instance, if you're asked to calculate the sum of the first N natural numbers during an interview:

- One candidate might use a loop:

```
int sum = 0;
for (int n = 1; n <= N; n++) {
    sum += n;
}
```

- Another candidate might use the formula: $Sum=N\times(N+1)/2$

The latter approach is more efficient and would likely be favored by the interviewer.

Familiarity with data structures like Hash Maps, Trees, Graphs, and algorithms equips you to solve problems effectively, much like a mechanic uses the right tools to fix a car. Interviewers seek candidates who can select and apply the best tools to address the challenges they are presented with. Understanding the characteristics of different data structures helps you make informed decisions in problem-solving.

Practical Use of DSA: If you enjoy solving real-world problems, DSA is invaluable.

Consider these examples:

- **Organizing Books in a Library:** Suppose you need to find a book on Data Science. You'd first go to the technology section, then look for the Data Science subsection. If the books were not organized in this way, it would be much more challenging to find the specific book. Data structures help organize information in a computer system, enabling efficient processing based on the provided input.

- **Managing a Playlist:** If you have a playlist of your favorite songs, you might arrange them in a particular order, such as by genre or mood. A **Queue** could be used to manage the order in which the songs play, ensuring that they play in the sequence you prefer.
- **Navigating a City:** When finding the shortest route between two locations in a city, you could use a **Graph** data structure to represent the roads and intersections, with algorithms like Dijkstra's or A* to find the optimal path.

These examples illustrate the importance of choosing the right data structure and algorithm for real-world problems, helping to solve them more efficiently.

Data Structures and Algorithms deepen your understanding of problems, allowing you to approach them more effectively and gain a better grasp of the world around you.

BIBLIOGRAPHY

- Board Infinity course
- Board Infinty Website
- Google